Mesh Blending

Yu-Shen Liu, Hui Zhang, Jun-Hai Yong, Pi-Qiang Yu, Jia-Guang Sun

Mesh blending

Yu-Shen Liu
Hui Zhang
Jun-Hai Yong
Pi-Qiang Yu
Jia-Guang Sun

Y.-S. Liu (✉) · H. Zhang · J.-H. Yong · J.-G. Sun
School of Software,
Tsinghua University, Beijing 100084, P.R. China
liuyushen00@mails.tsinghua.edu.cn,
{huizhang, yongjh}@mail.tsinghua.edu.cn

Y.-S. Liu · J.-G. Sun
Department of Computer Science and Technology,
Tsinghua University, Beijing 100084, P.R. China

P.-Q. Yu
School of Computer and Information Technology,
Beijing Jiaotong University, P.R. China
yupiqiang@tsinghua.org.cn

Abstract A new method for smoothly connecting different patches on triangle meshes with arbitrary connectivity, called mesh blending, is presented. A major feature of mesh blending is to move vertices of the blending region to a virtual blending surface by choosing an appropriate parameterization of those vertices. Once blending is completed, the parameterization optimization is performed to perfect the final meshes. Combining mesh blending with multiresolution techniques, an effective blending technique for meshes is obtained. Our method has several advantages: (1) the user can intuitively control the blending result using different blending radii, (2) the shape of cross-section curves can be adjusted to flexibly design complex models, and (3) the resulting mesh has the same connectivity as the original mesh. In this paper, some examples about smoothing, sharpening, and mesh editing show the efficiency of the method.

Keywords Mesh blending · Smoothing · Parameterization · Rolling ball

1 Introduction

With 3D scanners becoming the standard source for geometric data acquirement, triangle meshes have been the primary representations for reverse engineering, rapid prototyping, conceptual design, and simulation [2, 4, 18]. Mesh editing such as boolean operations, cut-and-paste editing, and deformation also becomes an active research area in computer graphics. Currently, it still is a challenge to flexibly and intuitively blend different patches in a given mesh. In this paper, we introduce a mesh blending technique based on a rolling ball. This technique is capable of producing desirable results.

Surface blending, whose goal is to create an intermediate surface that smoothly connects adjacent surfaces, is a way of constructing complex objects in geometric modeling. Surface blending is widely used for aesthetic, manufacturability, stress-concentration avoidance, and functional reasons [30, 34]. The blending operation has been widely studied for parametric and implicit surfaces. It is a nontrivial task to perform the operation on a mesh. The mesh blending technique presented in this paper allows the user to specify different blending radii to intuitively control the blending result and a few cross-section curves to flexibly design complex models.

1.1 Related work

Boolean operations often produce intersection curves from a set of original meshes. Continuity at intersection curves can be improved by blending or smoothing [37].

There are two popular smoothing approaches on triangle meshes: geometric filtering and geometric optimization [20]. Geometric filtering [9, 33] is designed to remove the high-frequency noise from an initial mesh. These techniques are isotropic, so shape features are usually diffused and lost. Feature-preserving smoothing has also been presented (e.g., [8, 10, 17]). However, geometric filtering lacks flexible control for shape and often leads to oversmoothing [26]. Geometric optimization is usually a computationally expensive task. Moreover, it also lacks local control for shape and often changes the connectivity of the original mesh. Generally, blending is accomplished under the direction of a user-specified distance and some complex profile curves. Therefore, the blending operator is more flexible when it comes to controlling the shape than the smoothing operator. Recently, several attempts have been made at finding blending methods for triangle meshes. Botsch et al. [6] describe a blending method based on the resampling technique for sophisticated modeling, which could change the mesh connectivity. Moreover, the shape of the resampling blending surface may be bad due to unknown center curves of rolling ball blending. Most recently, Museth et al. [24] presented a level set framework for blending an intersection curve produced by CSG boolean operations. However, since this method must convert models into volumetric representations, the details may be lost by blending.

On most blending methods of geometric modeling, these surface representations fall into two major categories: implicit and parametric surfaces [15, 35, 36]. Thus blending surfaces are also represented by parametric (or implicit) surfaces [3, 7]. However, it is rather difficult to smoothly connect several triangle meshes along common boundary curves using parametric (or implicit) surfaces. In addition, after some blending surfaces are added, the resulting models, including meshes and parametric (or implicit) surfaces, are difficult to maintain. Though blending surfaces can also be constructed by subdivision surfaces [19], most subdivision schemes need some given boundary curves, cross-boundary derivatives, and initial control nets. Therefore, it becomes difficult to perform the blending operation on triangle meshes using subdivision schemes.

To overcome the aforementioned difficulties, we present a mesh blending method based on a rolling ball that is common in parametric surface blending. The paper presented by Vida et al. [35] reviews all kinds of blending techniques that use parametric surfaces. One of the most popular methods is rolling ball blending (RBB) because of its simple geometric description and intuitive behavior [3, 7, 35]. A *blending surface* is an intermediate surface that smoothly connects two intersecting surfaces called *base surfaces*. Using the RBB method, the blending surface is generated by moving a ball in contact with two base surfaces. The trajectory of the ball's center is called a *spine curve*, while the common boundary of one base surface and the blending surface is called a *linkage curve*. The blending surface is part of a sweep surface constructed by sweeping a planar arc cross-section curve along the spine curve. In the general case, the intersection curve of the offsets of two base surfaces is first constructed as a spine curve. Next, the cross-section curve is defined as a rational quadratic curve between the two ball contact points. Finally, the blending surface is built by sweeping the cross-section curve.

1.2 Contributions

In this paper, the mesh blending method for smoothly connecting different patches on triangle meshes with arbitrary connectivity is presented. First, our method constructs closed blending curves and finds the trajectory of the rolling ball's center. Second, various cross-section curves are created for adjusting the blending shape, and then a virtual blending surface generated by sweeping these cross-section curves is designed. Third, we apply conformal parameterization to the blending region of the original mesh and optimize the shapes of these triangles in 2D parameter space. Finally, using the parameterization, we project vertices of the blending region onto the virtual blending surface such that the mesh blending is implemented successfully. The major contributions of our work are as follows.

- The mesh blending technique allows the user to easily control the blending result by a user-specified blending radius and also provides a desire mode of designing complex models by setting a few cross-section curves.
- An efficient CSC (constructing spine curve) algorithm is presented to find the trajectory of the rolling ball's center.
- A regressive method is used to eliminate self-intersection.
- An method for automatically splitting the blending region for generating parameterization is provided.
- The mesh blending operator can be widely used in applications such as smoothing, sharpening, and mesh editing.

The paper is organized as follows. Section 2 introduces some preliminary definitions. The method for finding the trajectory of the rolling ball is provided in Sect. 3. Section 4 constructs the virtual blending surface. Section 5 provides the method of parameterization and performs optimization. Some applications are presented in Sect. 6, and conclusions are given in Sect. 7.

2 Preliminaries

We assume that all triangle meshes considered in this paper are oriented manifolds. According to the definition

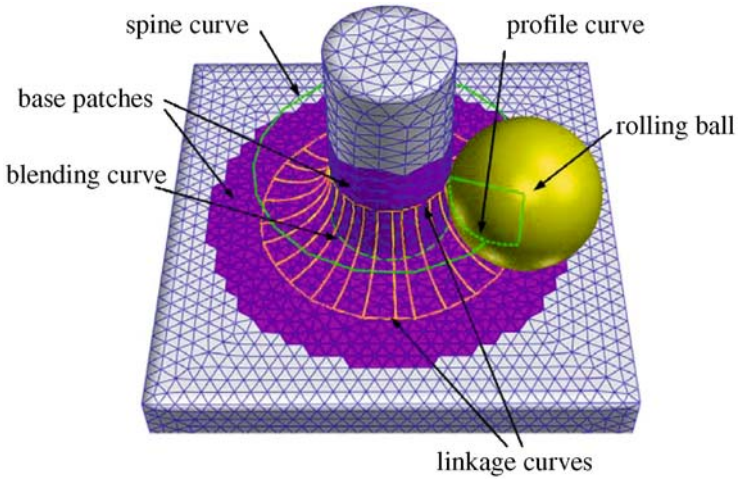

Fig. 1. The terminologies of mesh blending

of surface blending, we give the parallel definition of mesh blending based on RBB. A *patch* is a set of vertices and triangles on triangle meshes. It is also a part of triangle meshes. The operation of creating smooth transitions between two adjacent patches of triangle meshes is called *mesh blending*. A *blending patch* is an intermediate patch that smoothly connects two adjacent patches. One of two adjacent patches joined smoothly is called a *base patch*. The common intersection curve of two base patches is called the *blending curve*, which consists of a set of some *feature edges* whose dihedral angle formed by its two adjacent faces is larger than a given threshold. The trajectory of the rolling ball's center is called a *spine curve*, while the trajectories of the ball's contact points with base patches are called *linkage curves*. At each point of the spine curve, a cross-section curve that passes the ball's center and two contact points is called a *profile curve*. The *blending region* consists of all vertices and triangles between left and right linkage curves. Figure 1 illustrates the terminologies of mesh blending.

3 Constructing spine curve (CSC)

In practice, the blending surface based on RBB can be defined by sweeping the arc profile curves along the spine curve [7]. The spine curve of RBB can be defined as the intersection curve of two surfaces formed by offsetting base surfaces with a distance of the ball's radius. In this section, we show how to extract the blending curve and find the intersection curve of two offset base patches as the spine curve for triangle meshes.

Boolean operations often produce intersection curves and sharp features. In general, the user may specify the intersection curves and sharp features as the blending curves. In this paper, we assume that initial feature edges can be either extracted using a thinning algorithm [25] or interactively constructed in a similar "*backbones*" way [6]. Then these initial feature edges are chained together into a blending curve, as shown in Fig. 2a. For simplicity, we assume that the blending curve is closed. We can also extend our algorithm to the open blending curve when two endpoints are considered.

Mesh blending is a local operation because it is only applied in the neighborhood of the blending curve. Let r be the blending radius specified by the user. We denote the base patch by F_i and the corresponding offset patch by F_i^O, $i = 1, 2$. We define the regions influenced based on the distance $d = \alpha r$ to the blending curve as base patches F_1 and F_2. We typically use $\alpha = 1.75$. A larger value for α spends more time in intersecting two offset patches. In general, offset surface is defined as the locus of the points that are at a constant distance along the normal from the original surface [22]. This paper approximates the offset patch F_i^O of the base patch F_i by the following method.

1. Generate new vertices by moving all vertices of the base patch F_i along their normal directions with a distance r.
2. Construct the offset patch F_i^O based on these new vertices and the connectivity of the base patch F_i. The connectivity of offset patch F_i^O is consistent with the base patch F_i.

In this way, we can establish a one-to-one correspondence between the triangles of F_i^O and those of F_i. In Sect. 4.3, we will discuss the problem of self-intersection.

Once offsetting is finished, we will search for a continuous and closed curve from the intersection curve of two offset patches as the spine curve. The intersection problem of two triangle meshes has been discussed in many papers (e.g., [5, 27]). The task of mesh intersec-

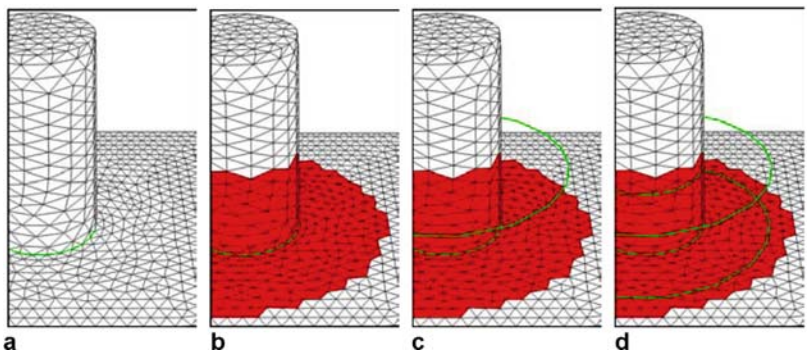

Fig. 2a–d. Examples to illustrate an overview of CSC procedure. **a** The (*green*) blending curve is extracted. **b** Find two (*red*) base patches along the blending curve. **c** Search for the spine curve through intersecting two offset base patches. **d** Find two linkage curves

tion reduces to computing the intersection of each pair of triangles from two meshes. Consequently, the intersection curve is constructed by connecting a series of triangle pair intersections. Each intersection pair relies on testing whether an edge of one triangle is intersected with another triangle. We use a recent intersection algorithm developed by Biermann et al. [5], which computes an intersection curve of two triangle meshes for approximating boolean operations on free-form solids. To accelerate the algorithm, we also combine a bounding box intersection test with a fast triangle–triangle intersection test [23]. Let $\boldsymbol{I}(t)$ be the intersection curve and $\boldsymbol{C}(t)$ the spine curve, where the parametric variable t is defined in an interval. In addition, we define two linkage curves by $\boldsymbol{C}_1(t)$ and $\boldsymbol{C}_2(t)$. In fact, $\boldsymbol{I}(t)$ is a set of all line segments in which each one is obtained by computing the triangle–triangle intersection. Note that one endpoint of each line segment is an *intersecting point* obtained by intersecting one edge with one triangle. So $\boldsymbol{I}(t)$ can be denoted by a list that stores a sequence of intersecting points $\{\boldsymbol{I}(t_i)\}$, where t_i is the parameter of the corresponding intersecting point. Each node of the list includes the intersecting triangle face, the intersecting edge, and the intersecting point's position. The data structure of a node is defined as

struct IntersectionPoint {
// intersection point of an edge and a face
Face **inter_face*;
// intersecting face of F_1^O *(or* F_2^O*)*
Edge **inter_edge*;
// intersecting edge of F_2^O *(or* F_1^O*)*
Point3D **inter_position*
// position of intersecting point
}.

In the case of the intersection of two edges, we choose one edge as *inter_edge* and one adjacent face of another edge as *inter_face*. The procedure for CSC is outlined as follows.

1. Construct the offset patch F_i^O that is at a constant distance r along the normal direction for every vertex of F_i $(i = 1, 2)$. F_i^O and F_i have the same connectivity.
2. Find the intersection curve $\boldsymbol{I}(t)$ between F_1^O and F_2^O, where $\boldsymbol{I}(t)$ may contain nonmanifold vertices that have no neighborhoods, discontinuous edges, and multiple loops. Each intersecting point of $\boldsymbol{I}(t)$ is stored in the foregoing node *IntersectionPoint*.
3. Search for a continue and closed loop from the intersection curve $\boldsymbol{I}(t)$ as the spine curve $\boldsymbol{C}(t)$ (Fig. 2c). Nonmanifold vertices, discontinuous edges, and open loops are removed from $\boldsymbol{I}(t)$. If multiple closed loops are found, the longest one is adopted as the result. Since $\boldsymbol{C}(t)$ is part of $\boldsymbol{I}(t)$, $\boldsymbol{C}(t)$ is also denoted by a list that stores a sequence of intersecting points $\{\boldsymbol{C}(t_k)\}$ $(k = 1, 2, \ldots, n)$. The parameter t_k can be computed by chord-length parameterization [29].
4. To determine one linkage curve $\boldsymbol{C}_i(t)$ $(i = 1, 2)$, project the spine curve $\boldsymbol{C}(t)$, i.e., each intersecting point $\boldsymbol{C}(t_k)$, onto the base patch F_i. First, the barycentric coordinate $\boldsymbol{P}_f$ of *inter_position* on *inter_face* and the parameter p_e of *inter_position* on *inter_edge* are calculated. Using $\boldsymbol{P}_f$ and p_e, the point's positions of the base face and edge are obtained (Fig. 2d).

4 Constructing the virtual blending surface

A blending surface based on RBB can be constructed using a sweeping algorithm described by Choi et al. [7]. Let $S(t, u)$ be the blending surface parameterized by (t, u), where t is a *sweeping parameter* and u is a *profile parameter*. The algorithm constructs the spine curve by fitting a cubic spline curve from the initial intersection points and denotes the resulting blending surface as a rational surface equation. We generalize this idea to approximate the sweep surface. In this section, we will construct a virtual blending surface approximated by a set of ruled surfaces.

The spine curve $\boldsymbol{C}(t)$ stores a sequence of intersecting points $\{\boldsymbol{C}(t_i)\}$, $i = 1, 2, \ldots, n$. For each point $\boldsymbol{C}(t_i)$, we can obtain a profile generated in Sect. 4.2. Let $\{\boldsymbol{T}_i(u) = S(t_i, u)\}$ and $i = 1, \ldots, n$ be a set of profiles corresponding to $\boldsymbol{C}(t_i)$. We can construct $(n-1)$ ruled surfaces instead of the sweep surface $S(t, u)$. For $t \in [t_i, t_{i+1}]$, the ith

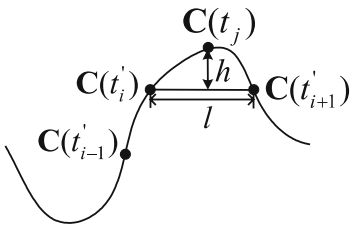

Fig. 3. Sampling points on spine curve

ruled surface is defined as

$$R_i(\hat{t}, u) = (1-\hat{t}) \cdot \boldsymbol{T}_i(u) + \hat{t} \cdot \boldsymbol{T}_{i+1}(u), \quad i = 1, \ldots, n-1, \tag{1}$$

where $\hat{t} = \frac{t-t_i}{t_{i+1}-t_i}$. These ruled surfaces will be used to compute the new vertex's positions on blending region through parameterization in Sect. 5.

During the process of generating the virtual blending surface, we can improve the blending surface quality and model more complicated shapes through two stages: sampling the spine curve and constructing complicated profile curves.

4.1 Point sampling

To improve the quality of blending surface, we should sample points on the spine curve $\boldsymbol{C}(t)$. A common sampling approach is to choose a few points while preserving the shape of the curve [28].

We present a simple and effective sampling method for the spine curve $\boldsymbol{C}(t)$, which contains a set of discretizing points. Our sampling method chooses dense points at the interval of large curvature and sparse points at the interval of small curvature. First, we get some initial points by sampling $\boldsymbol{C}(t)$ about a density function denoted by d_s. Let L be the total length of all edges on the spine curve and n the number of initial sampling. One simple choice that produces good results is to set d_s equal to L/n. We typically use $n = 40$. Then we will refine adjacent sampling points. For two initial adjacent sampling points, let h denote the chord height and l the chord length. Some points where h/l is greater than a threshold will be selected. Figure 3 illustrates the method of sampling the spine curve. $\{\boldsymbol{C}(t_i')\}$ denotes the initial adjacent sampling points about the density function d_s on the spine curve $\boldsymbol{C}(t)$, and $\boldsymbol{C}(t_j)$ is the refinement points for $t_i' < t_j < t_{i+1}'$.

4.2 Complicated profile curve

The referred profile curve is an arc curve. To construct a more complicated model, we can adjust the shape of the profile curve. The profile curve can be built with Hermite interpolating [6], where two points and tangents are given. For fixed $t = t_0$, we obtain three points $\boldsymbol{P} = \boldsymbol{C}(t_0)$, $\boldsymbol{P}_1 = \boldsymbol{C}_1(t_0)$, and $\boldsymbol{P}_2 = \boldsymbol{C}_2(t_0)$. The profile curve is part of

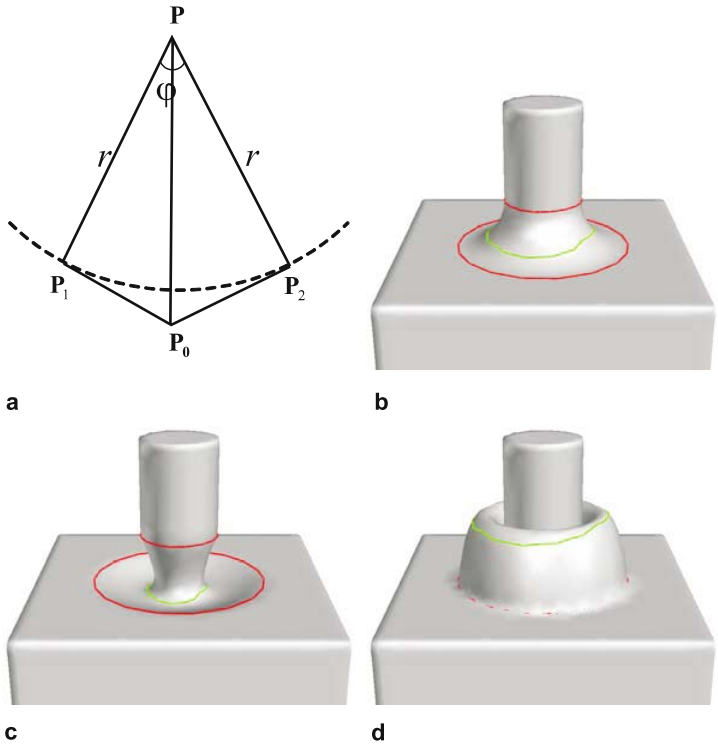

Fig. 4a–d. Results of three manipulations with different profile curves. **a** Construction of conic section control points. **b** Arc profile curve. **c** $\omega_0 = 2\cos(\varphi/2)$, $|\boldsymbol{P}_0 - \boldsymbol{P}| = 2.4 \times r$. **d** General profile curve

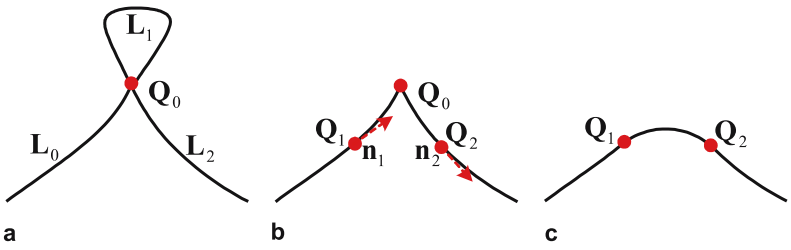

Fig. 5a–c. Curve procedure of our regressive method for eliminating self-intersection. **a** Initial self-intersection curve. **b** Delete self-intersection loop and regress self-intersection point. **c** Generate smooth curve

a circle with central $\boldsymbol{P}$ and radius r. Assume that φ is the angle opposite to the arc $\overset{\frown}{\boldsymbol{P}_1\boldsymbol{P}_2}$, as illustrated in Fig. 4a. If $\boldsymbol{P}_0$ is the intersecting point of two tangent lines of arc $\overset{\frown}{\boldsymbol{P}_1\boldsymbol{P}_2}$ at points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, we have

$$\boldsymbol{P}_0 - \boldsymbol{P} = \alpha(\boldsymbol{P}_1 - \boldsymbol{P} + \boldsymbol{P}_2 - \boldsymbol{P}). \tag{2}$$

Taking the dot product of Eq. 2 with $(\boldsymbol{P}_1 - \boldsymbol{P})$, we obtain

$$\begin{aligned}&(\boldsymbol{P}_0 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}) \\ &\quad = \alpha(\boldsymbol{P}_1 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}) + \alpha(\boldsymbol{P}_2 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}).\end{aligned}$$

As shown in Fig. 4a, we have that $(\boldsymbol{P}_0 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}) = r^2$, $(\boldsymbol{P}_1 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}) = r^2$, and $(\boldsymbol{P}_2 - \boldsymbol{P}) \cdot (\boldsymbol{P}_1 - \boldsymbol{P}) = r^2 \cdot \cos\varphi$, so

$$\alpha = \frac{1}{\cos\varphi + 1}.$$

Therefore, we have

$$\boldsymbol{P}_0 = \frac{\cos\varphi - 1}{\cos\varphi + 1} \cdot \boldsymbol{P} + \frac{1}{\cos\varphi + 1} \cdot (\boldsymbol{P}_1 + \boldsymbol{P}_2).$$

By taking $\boldsymbol{P}_1$, $\boldsymbol{P}_0$, and $\boldsymbol{P}_2$ as the control vertices of a conic profile curve, we define the NURBS representing the conic profile curve. If the weights are $\omega_0 = \cos(\varphi/2)$ and $\omega_1 = \omega_2 = 1$, we have an arc [29]

$$\boldsymbol{T}(u) = \frac{\sum_{i=0}^{2} \omega_i \boldsymbol{P}_i N_{i,p(u)}}{\sum_{i=0}^{2} \omega_i N_{i,p}(u)}, \tag{3}$$

where $\boldsymbol{P}_i$ are the control points and $N_{i,p}(u)$ are pth-degree B-spline basis functions defined on the knot vector $\boldsymbol{U} = \{0, 0, 0, 1, 1, 1\}$.

Figure 4 shows some examples with different profile curves. Control points of the conic profile curve are constructed in Fig. 4a. The influence of ω_0 to the blending curve of mesh is shown in Figs. 4b–d, where the green curve is the blending curve and the red curves are linkage curves.

4.3 Self-intersection

The blending surface may be self-intersecting if the radius of curvature of the spine curve is greater than the blending radius [21]. In general, the self-intersection surface is not suitable for applications. Although a number of algorithms [16, 21] could be used to detect self-intersection, no one presents the scheme of approximating the blending surface for eliminating self-intersection. A technique introduced by Botsch et al. [6] gave up the requirement that profile curves must be orthogonal to trajectories by a low-pass filter operator to the trajectories and cannot yield satisfying results for self-intersection in our blending. The low-pass filter operator contains a number of drawbacks: it lacks the local curve control and often leads to over-smoothing. Moreover, it is difficult to find a suitable low-pass filter operator for eliminating a large self-intersection loop.

In this section, we combine a regressive method with a low-pass filter operator for eliminating self-intersection, as illustrated in Fig. 5. Let $\boldsymbol{Q}_0$ be one self-intersection point on the spine curve. Once the self-intersection point is located, the spine curve $\boldsymbol{C}(t)$ should be split into three parts: the loop $\boldsymbol{L}_0$ before the self-intersection point, the self-intersection loop $\boldsymbol{L}_1$, and the remaining loop $\boldsymbol{L}_2$ (Fig. 5a). The self-intersection loop $\boldsymbol{L}_1$ must be deleted. Furthermore, $\boldsymbol{L}_0$ and $\boldsymbol{L}_2$ are merged into a new curve instead of the original spine curve $\boldsymbol{C}(t)$. Next, two regressive points $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are constructed by broadening at certain distances from $\boldsymbol{Q}_0$ (Fig. 5b). (The data structure for the intersecting point $\boldsymbol{Q}_0$ contains the intersecting face *inter_face*; we normally choose $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ on the boundary of *inter_face*.) Finally, a new curve segment is constructed by smoothly connecting regressive points $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$. Let $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$ be their corresponding tangent vectors. The new curve segment can be computed with Hermite interpolation satisfying endpoint conditions: $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, $\boldsymbol{n}_1$, and $\boldsymbol{n}_2$ [6]. In addition, we also smooth the final spine curve using a nonshrinkage Laplacian filter [33] (Fig. 5c). Figure 6 shows an example of eliminating self-intersection. The red curves are the spine curve and two linkage curves, and the blue curves are profile curves. An original mesh is shown in Fig. 6a, and the magnified view of the self-intersection region is shown in Fig. 6b. Figure 6c shows the initial spine curve containing one self-

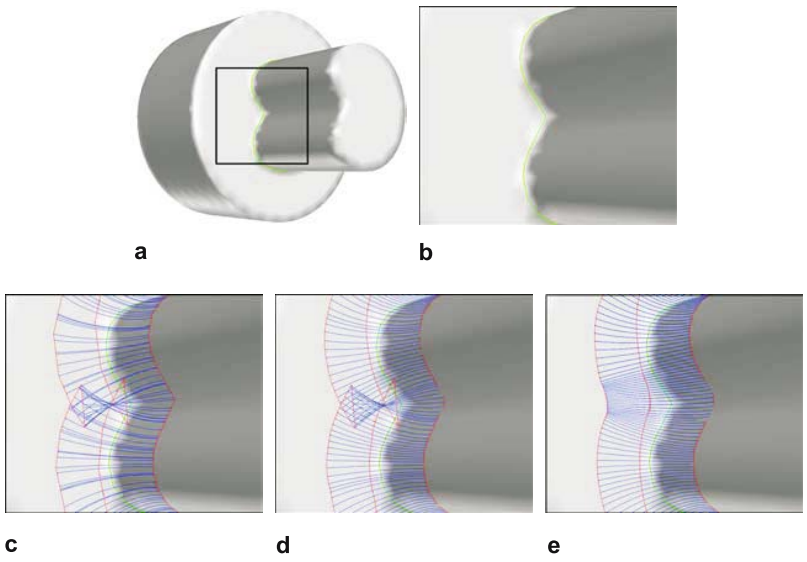

Fig. 6a–e. Example of overcoming self-intersection. **a** Initial mesh. **b** Self-intersection region. **c** Initial spine curve containing the self-intersection loop. **d** No-shrink Laplacian smoothing with a low-pass filter operator cannot delete the large self-intersection loop. **e** Smoothing according to our regressive method

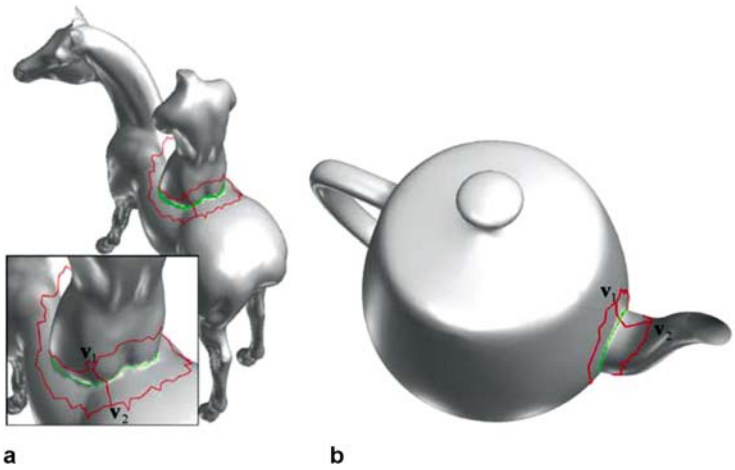

Fig. 7. Two examples of splitting the blending region

intersection loop. The low-pass filter operator used by Botsch et al. [6] cannot delete the large self-intersection loop, as shown in Fig. 6d. Our regressive method generates a more reasonable result, as shown in Fig. 6e.

A blending example of a boolean unit between the venus and a horse is illustrated in Fig. 7a, which is a self-intersection spine curve. The blending results are shown in Figs. 11 and 14c.

5 Parameterization

The goal of our blending method is to map the blending region to the virtual blending surface. Therefore, we need to find a map from the source to the target. In our case, the blending surface is approximated by sweeping the polygon profile curve. Note that the (u, v) domain of the blending surface is a quadrilateral (likely a rectangle), and most of the existing parameterization algorithms can fix the boundary on a square or any convenient rectangular region [11–13]. These methods are valid for disk-like patches. However, the blending region is topologically cylinderlike. Thus, we must split the blending region into disklike patches.

5.1 Splitting

In our case, it is not necessary to use a parameterization method that can compute natural boundaries. The method of mapping blending region is based on conformal parameterization [11] by fixing the boundary of the parameterization on a square. For this reason, we must split the blending region that has two boundaries. In [32], a pair of boundary vertices on the opposite boundaries is selected for preparing splitting, and the shortest path between these two vertices is constructed. In the shortest path algorithm,

the weight of each edge is set to its length. The shortcoming of the shortest path algorithm is that two boundary vertices must first be chosen by the user. Our splitting method can automatically produce these splitting vertices, which are two intersecting points between a *splitting plane* and two linkage curves $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$. Figure 7 shows two examples where our splitting algorithm is performed. The procedure for splitting is outlined as follows.

1. Construct a splitting plane defined by one profile curve, for example the longest arc profile curve. Intersect the splitting plane with two linkage curves $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, and calculate two intersection points on them.
2. Find the closest vertices $\boldsymbol{v}_1$ on $\boldsymbol{C}_1$ and $\boldsymbol{v}_2$ on $\boldsymbol{C}_2$ with intersection points.
3. Find the shortest path consisting of edges that are a subset of the blending region between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. The algorithms for finding the exact shortest path on a mesh usually involve high time and space costs. In this section, the Dijkstra's algorithm for finding the approximate shortest path is used.
4. Copy the edges on the shortest path, and split the blending region into a rectanglelike patch. Rebuild the mesh connectivity from the blending region, where four corner vertices ($\boldsymbol{v}_1$, $\boldsymbol{v}_2$ and their copy $\boldsymbol{v}_1'$, $\boldsymbol{v}_2'$) of the rectanglelike patch are obtained.

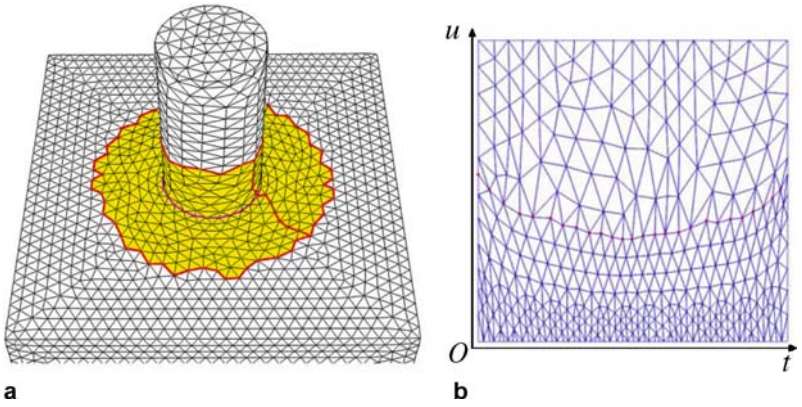

Fig. 8a,b. Conformal parameterization. **a** Original blending region of triangle mesh (*yellow*). **b** Its parameterization. Note that the *red points* on the parameterization region are vertices on the blending curve

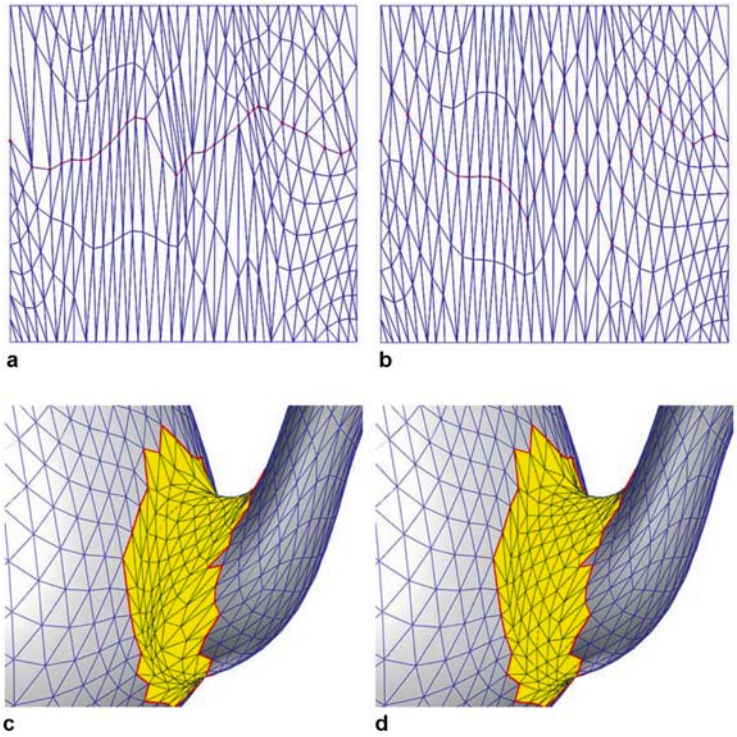

Fig. 9a–d. Optimize the blending (*yellow*) patch between teapot body and spout. **a** Initial parameterization. **b** Laplacian smoothing optimization in 2D parameter space. **c** Patch corresponding to **a** in 3D space. **d** Patch corresponding to **b** in 3D space

During the parameterization process, we advocate for the conformal parameterization as introduced in [11], since it minimizes the distortion of different intrinsic measures of the original mesh. We first fix boundary positions on the 2D domain and set rectangle's corner vertices which correspond to rectangle-like patches corner vertices ($\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_1^{'}, \boldsymbol{v}_2^{'}$). By solving a simple and sparse linear system, this parameterization automatically provides an angle-preserving mapping. Hence, we construct the map between the blending region and the virtual blending surface through parameterization. Next, the new position of every vertex on the blending region will be computed. For the two fixed parameters t_0 and u_0, the 3D point $S(t_0, u_0)$ is computed using Eq. 1. Finally, we successfully finish the blending operation for the blending curve of the mesh through projecting all vertices of the blending region onto the virtual blending surface. Figure 8 shows an example of splitting and conformal parameterization.

5.2 Optimization

For triangle meshes, optimization has two different aspects: *outer fairness* and *inner fairness* [31]. Not only should the shape of the blending surface be considered in outer fairness, but the quality of the mesh's triangles should also be improved in inner fairness.

All vertices of the mesh in the blending region "slide" along the virtual smooth blending surface when the blending radius is given. Our blending method naturally satisfies outer fairness in the interior of the blending region. Depending on Taubin smoothing [33], we perform a local optimization for boundary vertices of the blending region, which reduces unnatural deformations. In practice, since these boundary vertices are on the boundary of the virtual blending surface, only a few iteration steps are needed.

We use a weighted Laplacian smoothing to optimize the shape of triangles in the 2D parameter space. It is efficient for inner fairness [1, 20] and does not change the mesh connectivity. There are many different ways for weights to be chosen. We perform an optimization by choosing the weights used in [1]. Figure 9 shows an example of optimization.

6 Applications

The blending method presented in this paper provides a powerful mesh-modeling toolbox. In this section we demonstrate the effectiveness of our method by considering a variety of applications, including smoothing, sharpening, and editing.

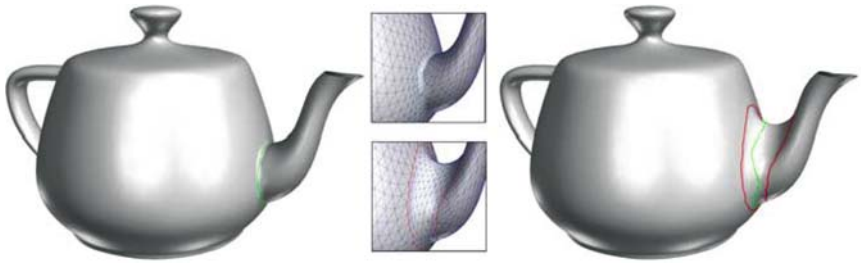

Fig. 10. Smoothing the sharp feature of a teapot between body and spout

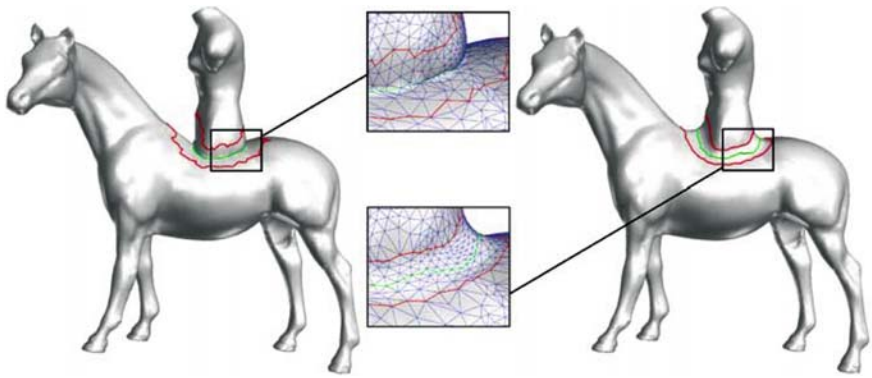

Fig. 11. Smoothing the intersection curve of boolean unit between the venus and a horse

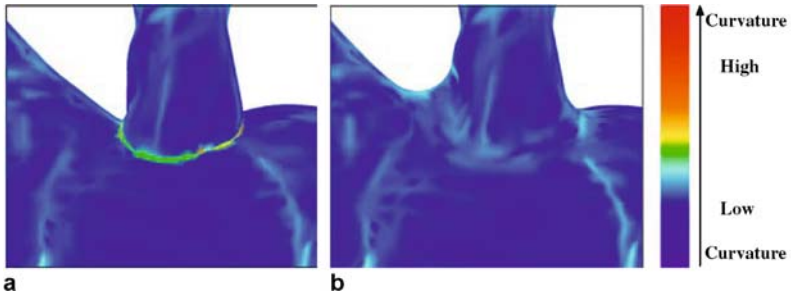

Fig. 12a,b. The color images of mean curvature show the successive smoothing by blending. **a** Original mesh. **b** Blending mesh

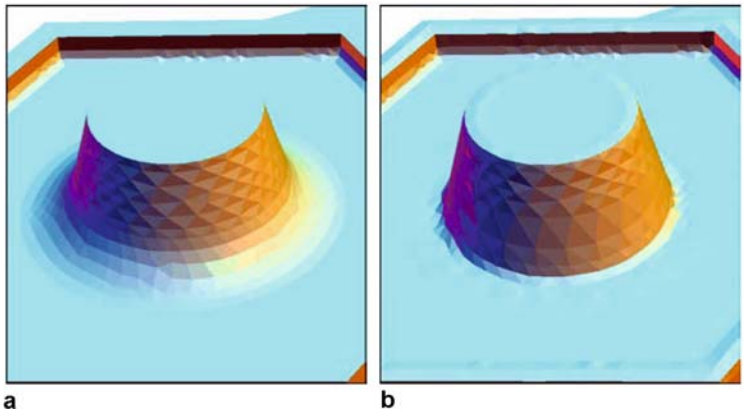

Fig. 13a,b. The rounded feature is sharpened by setting the sharpening profile curve. **a** Original mesh. **b** Sharpening result

6.1 Smoothing

Mesh blending can perform the smoothing operation for meshes while keeping the connectivity. Figure 10 illustrates an example of smoothing the sharp feature of a teapot between body and spout, and Fig. 11 shows the procedure of smoothing the intersection curve of the boolean unit between the venus and a horse. The original meshes are shown on the left, while the resulting meshes are shown on the right. We regard the sharp feature and the intersection curve as blending curves, and our mesh blending method is applied with a radius $r = 4cm$ and arc profile curves. The blending results on the right are geometrically smoothing. Figure 12 shows the curvature color images of the example as shown in Fig. 11. Note that the curvature is rescaled by blending, and the resulting mesh is smooth on the intersection curve.

6.2 Sharpening

In product design, feature characteristics are frequently changed. For example, in computational fluid dynamic (CFD) simulations, it is often necessary to vary the radius of RBB for verifying the impact on the overall aerodynamics [6]. Sharpening provides the opposite operation to smoothing. A rounded feature can be sharpened by setting the blending profile curve to the sharpening profile. Figure 13 illustrates the result of the sharpening operator. The original mesh with a rounded feature is shown in Fig. 13a, and the sharpening result is shown in Fig. 13b.

6.3 Mesh editing

Mesh editing includes boolean operations, cut-and-paste editing, deformation, and other techniques. Those techniques are often applied to obtain new models from a set of original meshes [37]. Moreover, mesh editing, e.g., boolean operations and cut-and-paste editing, may produce intersection curves. Continuity at intersection curves can be improved by blending. Our technique is capable of producing desirable blending results with various radii and a few cross-section curves. Figure 14 shows the blending results after boolean operations. In Fig. 14a,b, a new creature is produced by union of five meshes. Figure 14c shows the results of mesh blending with various radii and the arc profile curve. Figure 15 shows the application of complicated profiles to a teapot. Mesh blending can also be applied to other intersection curves produced by mesh editing, such as cut-and-paste editing, detail editing, and deformation.

By combining our algorithm with multiresolution techniques, we obtain an effective editing operation, i.e., mesh blending. We build a multiresolution mesh pyramid for large meshes using the algorithm presented by Guskov et al. [14] and only perform mesh blending at the coarsest resolution. The time is reported on a Pentium IV 1.70-GHz processor with 256M RAM. The execution time does not include time for loading meshes. For the teapot shown in Fig. 10, there are 8480 triangles on the fine mesh. First, we give a blending radius 4.5 cm and then find two base patches with 1423 triangles.

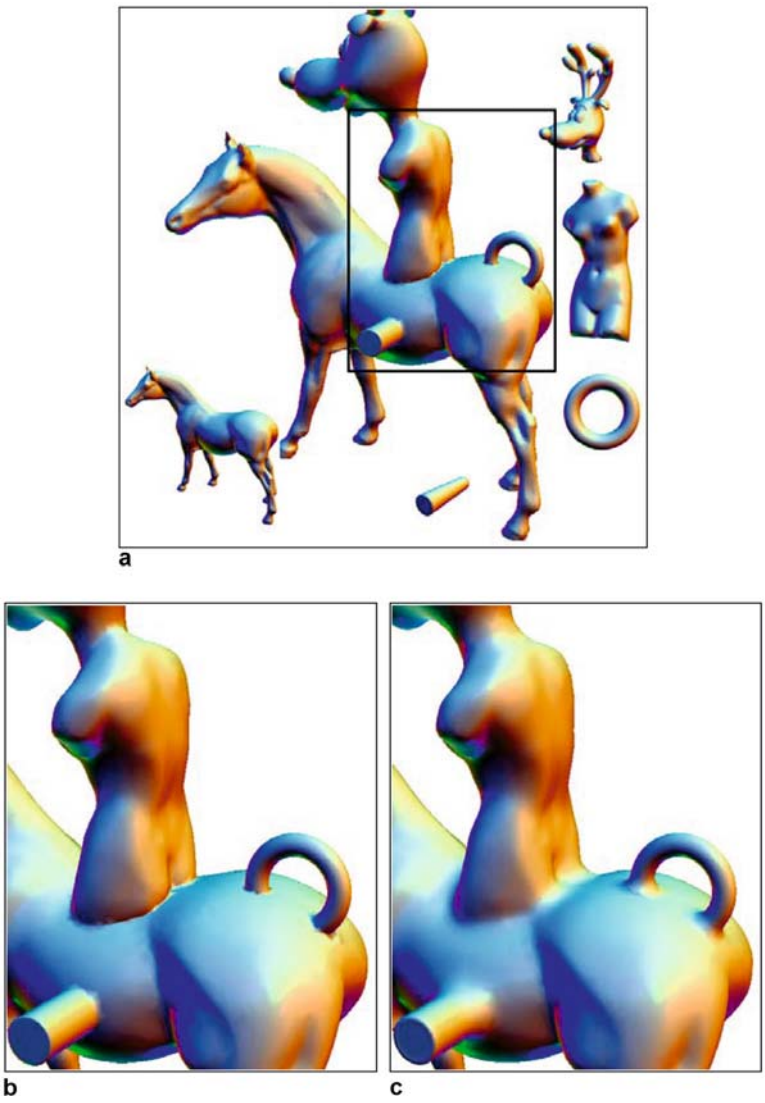

Fig. 14a–c. Applying mesh blending to intersection curves of boolean operations. **a** Union of five meshes. **b** Magnified view of **a**. **c** Resulting mesh with multi-step blending

Blending the fine mesh takes 1.515625 s. Comparatively, we start with the same model whose magnified view is shown in Fig. 16a and compute a coarsest mesh down to 1689 triangles, as shown in Fig. 16b. Through the blending curve as shown in Fig. 16c and the given radius, we find two base patches with 275 triangles. The blending operation for the coarse mesh shown in Fig. 16d takes 0.093750 s. Reconstruction is shown in Fig. 16e. So far the interactive blending operation for the complex mesh is finished.

7 Conclusions

We have presented a mesh blending method for smoothly connecting different patches on triangle meshes with arbitrary connectivity. The major idea is to move the vertices of the blending region to the virtual blending surface. Its major advantages are that it allows the user to specify different blending radii for directly controlling the blending shape and modify the shape of the profile curve for complex designs. Since our technique is a local operation, it is efficient for large meshes. We have also shown the efficiency of the method for some complicated meshes in smoothing, sharpening, and mesh editing.

The major drawback in our current implementation is that we map vertices of the blending region to the virtual blending surface by using a conformal parameterization method. This restricts the blended shapes and could fail in blending some complex blending regions with multiple holes. This could be improved by stitching together the virtual blending surface with the original mesh instead of by using the parameterization method. In the future we plan to extend the presented method to merge two or more separate models.

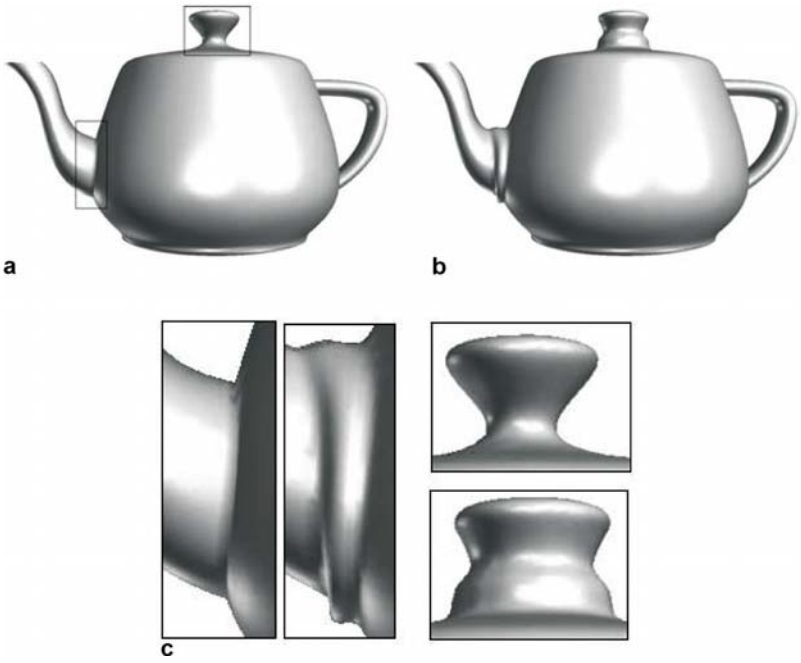

Fig. 15a–c. Blending with complicated profiles. **a** Original teapot. **b** Blending result with complicated profiles. **c** Magnified view of details

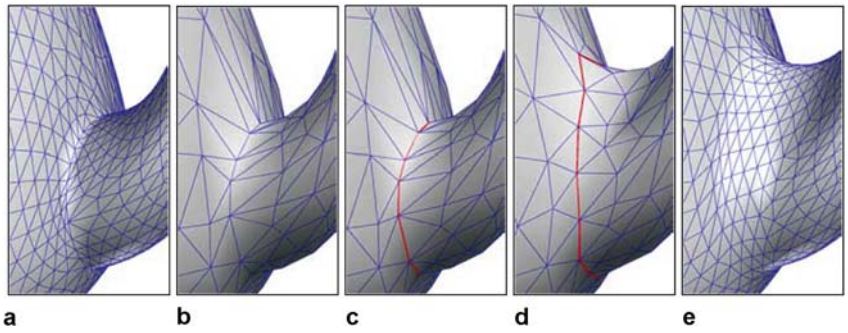

Fig. 16. Blending teapot between body and spout with multiresolution editing

Acknowledgement We would like to thank Chang-Cai Zhu, Bin Wang, Yi-Jun Yang, and Jie-Hui Gong for their valuable comments during our work. The authors appreciate the comments and suggestions of the anonymous reviewers. The research was supported by the Chinese 973 Program (2004CB719400) and the National Science Foundation of China (60403047). The third author was supported by a grant from the Foundation for Authors of National Excellent Doctoral Dissertations of PR China (200342).

References

1. Alliez P, Meyer M, Desbrun M (2002) Interactive geometry remeshing. In: Proceedings of SIGGRAPH'02, pp 347–354
2. Amenta N, Bern M (1999) Surface reconstruction by Voronoi filtering. Discrete Comput Geom 22(4):481–504
3. Barnhill RE, Farin GE, Chen Q (1993) Constant-radius blending of parametric surfaces. In: Farin GE, Hagen H, Noltemeier H (eds) Geometrie modeling. Springer, Berlin Heidelberg New York, pp 1–20
4. Bernardini F, Mittleman J, Rushmeier H, Silva C, Taubin G (1999) The ball-pivoting algorithm for surface reconstruction. IEEE Trans Visual Comput Graph 5(4):349–359
5. Biermann H, Kristjansson D, Zorin D (2001) Approximate boolean operations on free-form solids. In: Proceedings of SIGGRAPH'01, pp 185–194
6. Botsch M, Kobbelt L (2001) Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In: Proceedings of Eurographics'01, pp 402–410
7. Choi BK, Ju SY (1989) Constant-radius blending in surface modeling. Comput Aided Des 21(4):213–220
8. Clarenz U, Diewald U, Rumpf M (2000) Anisotropic geometric diffusion in surface processing. In: Proceedings of IEEE Visualization 2000, Salt Lake City, UT, pp 397–405
9. Desbrun M, Meyer M, Schröder P, Barr A (1999) Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of SIGGRAPH'99, pp 317–324

10. Desbrun M, Meyer M, Schröder P, Barr AH (2000) Anisotropic feature-preserving denoising of height fields and bivariate data. In: Proceedings of Graphics Interface 2000, pp 145–152
11. Desbrun M, Meyer M, Alliez P (2002) Intrinsic parameterizations of surface meshes. Comput Graph Forum 21(3):209–218
12. Floater M (1997) Parameterization and smooth approximation of surface triangulations. Comput Aided Geom Des 14(3):231–250
13. Floater M (2003) Mean value coordinates. Comput Aided Geom Des 20(1):19–27
14. Guskov I, Sweldens W, Schröder P (1999) Multiresolution signal processing for meshes. In: Proceedings of SIGGRAPH'99, pp 325–334
15. Hartmann E (2001) Parametric G^n-blending curves and surfaces. Visual Comput 17(1):1–13
16. Hermann T (1992) Rolling ball blends and self-intersections. In: Warren JD (eds) Curves and Surfaces in Computer Vision and Graphics III SPIE, pp 204–209
17. Jones TR, Durand F, Desbrun M (2003) Non-iterative, feature-preserving mesh smoothing. In: Proceedings of SIGGRAPH'03, pp 943–949
18. Kobbelt L, Botsch M (2000) An interactive approach to point cloud triangulation. In: Proceedings of Eurographics'00, pp 479–487
19. Levin A (1999) Combined subdivision schemes for the design of surfaces satisfying boundary conditions. Comput Aided Geom Des 16(5):345–354
20. Lévy B (2003) Dual domain extrapolation. In: Proceedings of SIGGRAPH'03, pp 364–369
21. Lukács G (1997) Differential geometry of G^1 variable radius rolling ball blend surfaces. Comput Aided Geom Des 15(6):585–613
22. Maekawa T (1999) An overview of offset curves and surfaces. Comput Aided Des 31(2):165–173
23. Möller T (1997) A fast triangle-triangle intersection test. J Graph Tools 2(2):25–30
24. Museth K, Breen DE, Whitaker RT, Barr AH (2002) Level set surface editing operators. In: Proceedings of SIGGRAPH'02, pp 330–338
25. Nomura M, Hamada N (2001) Feature edge extraction from 3D triangular meshes using a thinning algorithm. In: Proceedings of SPIE conference on vision geometry X, SPIE 4476:34–41
26. Ohtake Y, Belyaev A, Bogaevski I (2001) Mesh regularization and adaptive smoothing. Comput Aided Des 33(11):789–800
27. O'Rourke J (1994) Computational geometry in C. Cambridge University Press, Cambridge, UK
28. Park H, Kim K, Lee S-C (2000) A method for approximate NURBS curve compatibility based on multiple curve refitting. Comput Aided Des 32(4):237–252
29. Piegl L, Tiller W (1997) The NURBS book. Springer, Berlin Heidelberg New York
30. Rossignac JR, Requicha AAG (1984) Constant-radius blending in solid modeling. In: Proceedings of Computers in Mechanical Engineering, 3 July 1984, pp 65–73
31. Schneider R, Kobbelt L (2001) Geometric fairing of irregular meshes for free-form surface design. Comput Aided Geom Des 18(4):359–379
32. Shlafman S, Tal A, Katz S (2002) Metamorphosis of polyhedral surfaces using decomposition. Comput Graph Forum 21(3):219–228
33. Taubin G (1995) A signal processing approach to fair surface design. In: Proceedings of SIGGRAPH'95, pp 351–358
34. Várady T, Vida J, Martin RR (1989) Parametric blending in a boundary representation solid modeler. In: Handscomb, DC(eds) The mathematics of surfaces III. Oxford University Press, Oxford, UK, pp 171–197
35. Vida J, Martin RR, Várady T (1994) A survey of blending methods that use parametric surfaces. Comput Aided Des 26(5):341–365
36. Wu T, Zhou Y (2000) On blending of several quadratic algebraic surfaces. Comput Aided Geom Des 17(8):759–766
37. Yu Y, Zhou K, Xu D, Shi X, Bao H, Guo B, Shum HY (2004) Mesh editing with poisson-based gradient field manipulation. In: Proceedings of SIGGRAPH'04, pp 644–651





YU-SHEN LIU is a Ph.D. student in the Department of Computer Science and Technology at Tsinghua University, China. He received his B.Sc. in mathematics from Jilin University of China in 2000. His research interests are computer-aided design and computer graphics.

HUI ZHANG is an assistant professor in the School of Software at Tsinghua University, China. She received her B.Sc. and Ph.D. in computer science from the Tsinghua University of China in 1997 and 2003, respectively. Her research interests are computer-aided design and computer graphics.

JUN-HAI YONG is an associate professor in the School of Software at Tsinghua University, China. He received his B.Sc. and Ph.D. in computer science from Tsinghua University, China, in 1996 and 2001, respectively. He held a visiting researcher position in the Department of Computer Science at Hong Kong University of Science and Technology in 2000. He was a postdoctoral fellow in the Department of Computer Science at the University of Kentucky from 2000 to 2002. His research interests include computer-aided design, computer graphics, computer animation, and software engineering.

PI-QIANG YU is an assistant professor in the School of Computer and Information Technology at Beijing Jiaotong University, China. He received his B.Sc. and Ph.D. degrees in computational mathematics in 1997 and 2002 both from Dalian University of Technology of China, and he finished his postdoctoral research at Tsinghua University in 2004. His current research interests are computer graphics and computer-aided geometric design.

JIA-GUANG SUN is a professor in the Department of Computer Science and Technology at Tsinghua University, China. His research interests include computer graphics, computer-aided design, computer-aided manufacturing, product data management, and software engineering.